Computers and Mathematics with Applications 57 (2009) 184–194

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Quasi-self-stabilization of a distributed system assuming read/write atomicity

Ji-Cherng Lin *, Tetz C. Huang, Cheng-Zen Yang, Nathan Mou

Department of Computer Science and Engineering, Yuan-Ze University, Taiwan, ROC

ARTICLE INFO

Article history:
Received 30 October 2006
Accepted 21 February 2008

Keywords:
Computational models
Read/write atomicity
Composite atomicity
Self-stabilizing
Quasi-self-stabilizing
0-configurations
Mutual exclusion

ABSTRACT

Self-stabilizing systems of the Dolev type were first introduced by Dolev et al. in their famous paper in 1993. In contrast to self-stabilizing systems of the Dijkstra type, such self-stabilizing systems assume the read/write atomicity model instead of the composite atomicity model. In this paper, we introduce the notion of quasi-self-stabilizing systems of the Dolev type. A naturally-adapted version from Dijkstra's K-state mutual exclusion algorithm is employed to illustrate the new notion. The adapted algorithm is shown to be self-stabilizing if K is greater than or equal to $2n-1$, quasi-self-stabilizing but not self-stabilizing if K is less than $2n-1$ but greater than or equal to n, and not quasi-self-stabilizing if K is less than n.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

In 1974, Dijkstra first introduced the notion of self-stabilization in his pioneering paper [1]. After having been neglected for nearly a decade, the paper was drawn to public attention by Lamport in his invited address at PODC 1983 (cf. [2]). Since then, the research on self-stabilizing systems has flourished, and a great number of papers regarding self-stabilizing algorithms have been published. Most of these papers adopt Dijkstra's computational model, which is generally referred to as the central demon model.

1.1. The central demon model

Dijkstra's *central demon model* of computation (cf. [3,4]) of an algorithm in a distributed system has the following features:

(a) The algorithm running on each processor consists of one or more rules. Each rule is of the form

$$\textit{condition part} \rightarrow \textit{action part}.$$

The *condition part* (or *guard*) is a Boolean function over the states of the processor and its neighbors; the action part is an assignment of values to some of the processor's shared registers. If the condition part of a rule in a processor is evaluated as true, we say that the processor is *privileged* to execute the action part (or to *make a move,* or to *write*).

(b) At the initial configuration, if none of the processors is privileged, then the system is deadlocked. Otherwise, if a privileged processor exists, the *central demon* will randomly select exactly one among all the privileged processors to make a move, in a single atomic step. The local state of the selected processor thus changes, which in the meantime

* Corresponding address: Department of Computer Science and Engineering, Yuan-Ze University, 135 Yuan-Tung Road, Chung-Li 320, Taiwan, ROC. Tel.: +886 3 4638800x2354; fax: +886 3 4638850.
E-mail address: csjclin@saturn.yzu.edu.tw (J.-C. Lin).

0898-1221/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2008.02.052

results in the change of the global configuration of the system. The system will then repeat the above process again and again to change the global configuration as long as it does not encounter any deadlock situation. Thus, the behavior of the system under the action of the algorithm can be described by an *execution sequence* (or, simply, *execution*) $\Gamma = (\gamma_1, \gamma_2, \ldots)$ in which for any $i \geq 1$, γ_i represents a global configuration, and γ_{i+1} is obtained from γ_i after exactly one processor in the system makes the ith *move* $\gamma_i \rightarrow \gamma_{i+1}$.

Under this computational model, Dijkstra introduced the notion of self-stabilization (cf. [1,5,6]). According to Dijkstra, an algorithm is *self-stabilizing* if, regardless of any initial configuration of the system, any execution of the algorithm will lead the system to a legitimate configuration, and then let the system stay in the legitimate configuration (or some legitimate configurations) forever unless the system incurs a subsequent transient fault.

1.2. The Dolev model

One observes that Dijkstra's central demon model assumes the composite atomicity. A single move (or atomic step) by a processor consists of reading registers of all its neighbors, making internal computations and then rewriting its own register (or registers). In 1993, Dolev et al. introduced a new type of computational model in their famous paper [7]. Their model reflects more truthfully a real distributed system. Firstly, it assumes the *read/write atomicity*. Under such an assumption, each move (or atomic step) in the system consists of internal computations and either a single read operation or a single write operation. Secondly, it assumes that each processor in the system runs its own program indefinitely and at its own pace. Finally, it assumes that each processor in the system has a *program counter*. Thus the running of the program in a processor has to follow the order of instructions in the program according to the program counter. Under the Dolev model, the behavior of the system under the action of the algorithm can still be described by an execution sequence $\Gamma = (\gamma_1, \gamma_2, \ldots)$. As in Dijkstra's central demon model, in any configuration γ_i, a unique processor of the system is selected by the central demon to make the move $\gamma_i \rightarrow \gamma_{i+1}$, and thus change the system configuration to γ_{i+1}. However, due to the content of the algorithm and the way in which the algorithm is executed, the selection by the central demon is no longer random in a system of the Dolev type. In other words, any execution sequence of an algorithm in a system of the Dolev type has to obey certain restrictions (for instance, since every processor in the system runs its own program indefinitely, any execution of the algorithm under the Dolev model must be *fair*, i.e., every processor makes infinite number of moves in the execution). The definition for an algorithm to be *self-stabilizing under the Dolev model* is the same as that under Dijkstra's central demon model.

As one can easily see, self-stabilizing systems of the Dolev type are more realistic than those of the Dijkstra type. However, due to the intrinsic complication of the refined read/write atomicity and other restrictions imposed by the Dolev model, it is by no means a trivial matter to obtain a self-stabilizing system of the Dolev type. To the best of our knowledge, only a few papers regarding self-stabilizing algorithms under the Dolev model have been published in the past: Dolev et al. presented and proved the correctness of two self-stabilizing algorithms in [7,8], one of which is for the mutual exclusion problem for tree networks, and the other for the spanning tree problem for general networks. Also in [7,8], it is shown that a self-stabilizing mutual exclusion algorithm for general networks can be obtained by combining the above two algorithms. The combined algorithm thus obtained can serve as a "compiler" that can convert any central-demon-model self-stabilizing algorithm into a Dolev-model self-stabilizing algorithm (see [8, Section 4.1]). Collin and Dolev presented a self-stabilizing DFS algorithm in [9]. Recently, we have proposed a self-stabilizing center-finding algorithm in [10], and a self-stabilizing shortest-path-finding algorithm in [3].

Although all the above-mentioned algorithms, except the combined algorithm for the compiler, are naturally-adapted versions from self-stabilizing algorithms under Dijkstra's central demon model, the reader should not be led to believe that the naturally-adapted version from any Dijkstra-model self-stabilizing algorithm will automatically be Dolev-model self-stabilizing. This is not the case as we have shown in [3] using Dijkstra's mutual exclusion algorithm. Many other examples (e.g., the self-stabilizing maximal-independent-set algorithm in Shukla et al. [11]) can also be used to help clarify this issue. Moreover, when it happens that the naturally-adapted version from a Dijkstra-model self-stabilizing algorithm is not Dolev-model self-stabilizing, some further modification of the naturally-adapted version can be considered in order to come up with a desired Dolev-model self-stabilizing algorithm (although the success cannot always be guaranteed). We will pursue this direction in the near future.

1.3. Quasi-self-stabilization

In the following, we introduce the notion of quasi-self-stabilizing systems of the Dolev type, which reflects a phenomenon observed in our research on self-stabilizing systems of the Dolev type. First recall that a distributed system of the Dolev type is *self-stabilizing* if, starting with any global configuration, the system can converge to a legitimate configuration, and then stay in the legitimate configuration (or some legitimate configurations) thereafter unless it incurs a subsequent transient fault. Now we define that a system of the Dolev type is *quasi-self-stabilizing* if, starting with any global configuration that has every processor's program counter set to zero (such a global configuration will henceforth be abbreviated as a 0-*configuration*), the system can converge to a legitimate configuration, and then stay in the legitimate configuration (or some legitimate configurations) thereafter unless it incurs a subsequent transient fault. If a system starts with a 0-configuration, then every processor in the system will execute its own program from the very first instruction of the program. For a quasi-self-stabilizing system, the capability of recovering from a transient fault cannot be guaranteed, because some processors'

program counters may not be 0 after a transient fault, and thus these processors may not execute their programs from the first instruction after the transient fault. Nevertheless, if we consider the start-up situation, then, without any initialization, a quasi-self-stabilizing system is guaranteed to self-stabilize. This is because in the start-up situation, every processor's program counter has a 0-value, and so every processor will execute its own program from the very first instruction. From the definitions of the two types of self-stabilizing systems, one can see that a self-stabilizing system of the Dolev type is obviously a quasi-self-stabilizing system of the Dolev type. The converse, however, is not true, as the main results in this paper will clearly show.

1.4. Main results and related works

In this paper, the above notion of quasi-self-stabilizing systems of the Dolev type will be clarified. First, as in Dolev's book [8], we modify Dijkstra's K-state ($K \geq 2$) mutual exclusion algorithm into a version that can operate in a system of the Dolev type. The system is a ring of n nodes ($n \geq 2$). We will then show that the adapted algorithm is self-stabilizing if $K \geq 2n-1$, quasi-self-stabilizing but not self-stabilizing if $n \leq K < 2n-1$, and not quasi-self-stabilizing if $2 \leq K < n$.

Self-stabilizing algorithms for the mutual exclusion problem have been investigated in the past. One of the first three such algorithms in the literature, Dijkstra's K-state mutual exclusion algorithm, which adopts the central demon model, was presented in [1]. Dijkstra provided a proof in [5], showing that if $K \geq n$, his algorithm is self-stabilizing. In [8, Section 2.6], $K \geq n-1$ is shown to be a necessary and sufficient condition for Dijkstra's algorithm to be self-stabilizing. The proof provided there is under the assumption of the fair demon model (cf. [8, p. 16]). It is then pointed out that a ring of n nodes equipped with the naturally-adapted version of Dijkstra's K-state mutual exclusion algorithm, when operating under the Dolev model, can be viewed as just a ring of $2n$ nodes equipped with Dijkstra's algorithm operating under the central demon model. From this viewpoint and the above necessary and sufficient condition for Dijkstra's algorithm to be self-stabilizing under the fair demon model, it follows immediately that a ring of n nodes equipped with the adapted algorithm is self-stabilizing under the Dolev model if and only if $K \geq 2n-1$.

The first main result of this paper, which is to be obtained in Section 3, is exactly the same as the last result above. However, unlike Dolev's proof in [8], our proof deals directly with the complexity resulting from the refined read/write atomicity, instead of transforming a problem under the read/write atomicity model into a problem under the central demon model. Mainly due to such a difference, our proof can be modified to obtain the quasi-self-stabilization property of the adapted algorithm, the second main result of this paper, in Section 4.

1.5. Organization of the paper

The rest of this paper is arranged as follows. In Section 2, Dijkstra's K-state mutual exclusion algorithm is recalled, and an adapted version from it is proposed. The legitimate configurations are defined, and four types of legitimate configurations are emphasized. In Section 3, the adapted algorithm is shown to be self-stabilizing under the Dolev model if and only if $K \geq 2n-1$. In Section 4, the proof for the result in Section 3 is modified to show that the adapted algorithm is quasi-self-stabilizing under the Dolev model if and only if $K \geq n$. Finally in Section 5, some short remarks conclude this paper.

2. The algorithm and the legitimate configurations

Consider a distributed system whose underlying topology is a ring of n ($n \geq 2$) nodes. Each node represents a processor in the system. Nodes are numbered from 0 to $n-1$ in order. For each $i \in \{0, 1, 2, \ldots, n-1\}$, the edge between node i and node $i+1 \bmod n$ represents the unidirectional link between the two processors. Since the computational model employed here is of the Dolev type, the read/write atomicity is assumed. Thus, for our purpose, let each node i in the system maintain a shared register S_i (cf. Fig. 1), in which node i writes and from which node $i+1 \bmod n$ reads. The register is serializable with respect to read and write operations. Let node i also maintain a local variable r_i (cf. Fig. 1), in which node i stores the value that it reads from the register S_{i-1} of the neighbor $i-1 \bmod n$. The values of S_i's and r_i's are in the range $\{0, 1, 2, \ldots, K-1\}$, where $K \geq 2$ is a positive integer. A naturally-adapted version from Dijkstra's K-state mutual exclusion algorithm, for the above system, is as follows.

Algorithm 1.

{*For node 0*}
1. *repeat forever*
2. *read* $(r_0 := S_{n-1})$
3. *if* $S_0 = r_0$ *then* $write(S_0 := S_0 + 1 \bmod K)$ *end if*
4. *end repeat*

{*For node* $i \neq 0$}
1. *repeat forever*
2. *read*$(r_i := S_{i-1})$
3. *if* $S_i \neq r_i$ *then* $write(S_i := r_i)$ *end if*
4. *end repeat*

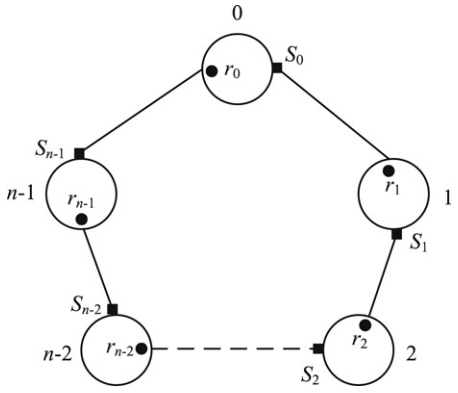

Fig. 1. The structure of a ring system of n nodes that is equipped with Algorithm 1 and assumes the Dolev model.

For each node, if the guard condition in statement 3 in its program is evaluated as true, then the node is said to be *privileged*. For the purpose of mutual exclusion, a global configuration of the system is said to be a *legitimate configuration* if in that configuration, there is at most one node privileged. One can easily check that the following four types of global configurations are legitimate configurations. (Note that in these configurations, $C \in \{0, 1, \ldots, K-1\}$ and $i \in \{1, 2, \ldots, n-1\}$.)

Type 1	r_0	S_0	r_1	S_1	$\cdots$	r_{n-1}	S_{n-1}		
	C	C	C	C	$\cdots$	C	C		
Type 2	r_0	S_0	$\cdots$	S_{i-1}	r_i	S_i	$\cdots$	r_{n-1}	S_{n-1}
	C	$C+1$	$\cdots$	$C+1$	C	C	$\cdots$	C	C
Type 3	r_0	S_0	r_1	$\cdots$	r_i	S_i	$\cdots$	S_{n-1}	
	C	$C+1$	$C+1$	$\cdots$	$C+1$	C	$\cdots$	C	
Type 4	r_0	S_0	r_1	S_1	$\cdots$	r_{n-1}	S_{n-1}		
	C	$C+1$	$C+1$	$C+1$	$\cdots$	$C+1$	$C+1$		

In the following sections, we shall present correctness proofs. To facilitate the presentations in these proofs, we hereby define a few expressions. Suppose t is a time instant and P is a predicate. If P is true right after the time instant t, then we express it as "P at t^+"; similarly, if P is true right before t, then we express it as "P at t^-"; if P at both t^+ and t^-, then we express it as "P at t". For instance, if, at a certain time instant t, node 0 in the above system executes the write operation "write ($S_0 := S_0 + 1 \bmod K$)" and changes the value of S_0 from 2 to 3, then $S_0 = r_0 = 2$ at t^-, and $S_0 = 3$ at t^+. However, if node 0 has $S_0 = 2$ at t^- and does not execute the write operation at t, then $S_0 = 2$ at t^+ and hence $S_0 = 2$ at t.

3. Self-stabilization of Algorithm 1

In order to claim that the system equipped with Algorithm 1 is self-stabilizing under the Dolev model, and solves the mutual exclusion problem, we are required to show that starting with any configuration, Algorithm 1 can ensure that (1) the system eventually converges to a legitimate configuration and then stay in legitimate configurations forever (thus there will always be no more than one node privileged); and (2) every node in the system will be privileged infinitely many times.

Lemma 1. *In any execution of Algorithm 1, node 0 writes infinitely many times.*

Proof. Suppose not. Then there exists an execution of Algorithm 1 and a time instant t_1 such that in this execution, node 0 never writes after t_1. Hence the value of S_0 never changes after t_1. Let τ_1 be the first time instant in the time interval (t_1, ∞) at which node 1 reads $r_1 = S_0$. Then we have $r_1 = S_0$ at τ_1^+. (1) If $S_1 = r_1$ at τ_1^+, then S_0, r_1 and S_1 are all equal and never change in (τ_1, ∞). (2) If $S_1 \neq r_1$ at τ_1^+, then there exists a $\tau_2 > \tau_1$ such that node 1 writes at τ_2. Thus $S_1 = r_1$ at τ_2^+, and S_0, r_1 and S_1 are all equal and never change in (τ_2, ∞). From (1) and (2) above, we can see that in any case, there exists a $t_2 > t_1$ such that $S_0 = S_1 = r_1$ after t_2. For the same reason, there exists a $t_3 > t_2 > t_1$ such that $S_0 = S_1 = S_2 = r_1 = r_2$ after t_3. Arguing in this manner, we will eventually get a $t_n > t_1$ such that $S_0 = \cdots = S_{n-1} = r_1 = \cdots = r_{n-1}$ after t_n. Then, in view of Algorithm 1, there exists a $t_{n+1} > t_n$ such that node 0 reads $r_0 = S_{n-1}$ at t_{n+1}. Thus $r_0 = S_{n-1} = S_0$ at t_{n+1}^+. But then, in view of Algorithm 1, there must exist a $t_{n+2} > t_{n+1} > t_1$ such that node 0 writes at t_{n+2}, which contradicts the definition of t_1. Thus the lemma is proved. ■

Lemma 2. *If $K \geq 2n-1$, then in any execution of Algorithm 1, the system will eventually converge to a legitimate configuration of Type 2.*

Proof. Let 0 be the initial time instant. Without loss of generality, we assume that $S_0 = 0$ at 0. Since node 0 can write infinitely many times in view of Lemma 1, we let $t_{01}, t_{02}, \ldots, t_{0(2n)}$, with $0 < t_{01} < t_{02} < \cdots < t_{0(2n)}$, be the first $2n$ instants at which node 0 executes write operations so that S_0 changes to $1, 2, \ldots, 2n, \mod K$, in sequence. Thus at the instants $t_{01}, t_{02}, \ldots, t_{0(2n)}$, $r_0 = 0, 1, \ldots, 2n-1, \mod K$, in sequence. Hence there exist $v_2, v_3, \ldots, v_{2n}$, with $t_{01} < v_2 < t_{02} < \cdots < t_{0(2n-1)} < v_{2n} < t_{0(2n)}$, such that at the instants $v_2, v_3, \ldots, v_{2n}$, node 0 executes read operations so that r_0 changes to $1, 2, \ldots, 2n-1, \mod K$, in sequence, and hence at the instants $v_2, v_3, \ldots, v_{2n}$, $S_{n-1} = 1, 2, \ldots, 2n-1, \mod K$, in sequence. Hence there exist $u_3, u_4, \ldots, u_{2n}$, with $v_2 < u_3 < v_3 < \cdots < v_{2n-1} < u_{2n} < v_{2n}$, such that at the instants $u_3, u_4, \ldots, u_{2n}$, node $n-1$ executes write operations so that S_{n-1} changes to $2, 3, \ldots, 2n-1, \mod K$, in sequence. Thus there exist $t_{(n-1)3}, t_{(n-1)4}, \ldots, t_{(n-1)(2n)}$, with $t_{01} < t_{(n-1)3} < t_{(n-1)4} < \cdots < t_{(n-1)(2n)} < t_{0(2n)}$, such that at the instants $t_{(n-1)3}, t_{(n-1)4}, \ldots, t_{(n-1)(2n)}$, node $n-1$ executes write operations so that S_{n-1} changes to $2, 3, \ldots, 2n-1, \mod K$, in sequence (here, simply let $t_{(n-1)j} = u_j$). Hence at the instants $t_{(n-1)3}, t_{(n-1)4}, \ldots, t_{(n-1)(2n)}$, $r_{n-1} = 2, 3, \ldots, 2n-1, \mod K$, in sequence. Therefore there exist $q_4, q_5, \ldots, q_{2n}$, with $t_{(n-1)3} < q_4 < t_{(n-1)4} < q_5 < t_{(n-1)5} < \cdots < t_{(n-1)(2n-1)} < q_{2n} < t_{(n-1)(2n)}$, such that at the instants $q_4, q_5, \ldots, q_{2n}$, node $n-1$ executes read operations so that r_{n-1} changes to $3, 4, \ldots, 2n-1, \mod K$, in sequence, and hence at the instants $q_4, q_5, \ldots, q_{2n}$, $S_{n-2} = 3, 4, \ldots, 2n-1, \mod K$, in sequence. Hence there exist $l_5, l_6, \ldots, l_{2n}$, with $q_4 < l_5 < q_5 < \cdots < q_{2n-1} < l_{2n} < q_{2n}$, such that at the instants $l_5, l_6, \ldots, l_{2n}$, node $n-2$ executes write operations so that S_{n-2} changes to $4, 5, \ldots, 2n-1, \mod K$, in sequence. Thus there exist $t_{(n-2)5}, t_{(n-2)6}, \ldots, t_{(n-2)(2n)}$, with $t_{(n-1)3} < t_{(n-2)5} < t_{(n-2)6} < \cdots < t_{(n-2)(2n)} < t_{(n-1)(2n)}$, such that at the instants $t_{(n-2)5}, t_{(n-2)6}, \ldots, t_{(n-2)(2n)}$, node $n-2$ executes write operations so that S_{n-2} changes to $4, 5, \ldots, 2n-1, \mod K$, in sequence (here, simply let $t_{(n-2)j} = l_j$).

Arguing in the same manner as above, we will finally get $t_{(n-1)3}, t_{(n-2)5}, \ldots, t_{2(2n-3)}, t_{1(2n-1)}, t_{1(2n)}, t_{2(2n)}, \ldots, t_{(n-1)(2n)}$, with $t_{01} < t_{(n-1)3} < t_{(n-2)5} < \cdots < t_{2(2n-3)} < t_{1(2n-1)} < t_{1(2n)} < t_{2(2n)} < \cdots < t_{(n-1)(2n)} < t_{0(2n)}$, such that for any $i = 1, 2, \ldots, n-1$, node i executes a write operation at $t_{i(2n)}$ so that S_i changes to $2n-1 \mod K$, and at the instants $t_{1(2n-1)}$ and $t_{1(2n)}$, $r_1 = 2n-2 \mod K$ and $2n-1 \mod K$, respectively. Hence there exists a t', with $t_{1(2n-1)} < t' < t_{1(2n)}$, such that at the instant t', node 1 executes a read operation so that r_1 changes to $2n-1 \mod K$, and hence at the instant t', $S_0 = 2n-1 \mod K$.

Summarizing all the above, we have

(1) $t_{01} < t_{1(2n-1)} < t' < t_{1(2n)} < t_{2(2n)} < \cdots < t_{(n-1)(2n)} < t_{0(2n)}$
(2) $S_0 = 2n-1 \mod K$ at t'
(3) $1, 2, \ldots, 2n-1, \mod K$, are all distinct (since $K \geq 2n-1$ by assumption)
(4) $S_0 = 1, 2, \ldots, 2n-1$ in $(t_{01}, t_{02}), (t_{02}, t_{03}), \ldots, (t_{0(2n-1)}, t_{0(2n)})$, in sequence (by the definitions of $t_{01}, t_{02}, \ldots, t_{0(2n)}$)
(5) For any $i = 1, 2, \ldots, n-1$, node i executes a write operation at $t_{i(2n)}$ so that S_i changes to $2n-1 \mod K$, and hence $r_i = 2n-1$ at $t_{i(2n)}$
(6) $S_0 = 2n-1 \mod K$ in $(t_{0(2n-1)}, t_{0(2n)})$. (We single this condition out from those in (4) above for presentation's sake.)

From (1), (2), (3) and (4), we get $t' \in (t_{0(2n-1)}, t_{0(2n)})$, and hence $t_{0(2n-1)} < t' < t_{1(2n)} < t_{2(2n)} < \cdots < t_{(n-1)(2n)} < t_{0(2n)}$. This, together with (5) and (6), implies $S_1 = r_1 = 2n-1$ in $(t_{1(2n)}, t_{0(2n)})$, $S_2 = r_2 = 2n-1$ in $(t_{2(2n)}, t_{0(2n)})$, ..., and $S_{n-1} = r_{n-1} = 2n-1$ in $(t_{(n-1)(2n)}, t_{0(2n)})$, in view of Algorithm 1. Thus we have $S_1 = S_2 = \cdots = S_{n-1} = r_1 = r_2 = \cdots = r_{n-1} = 2n-1 \mod K$ in $(t_{(n-1)(2n)}, t_{0(2n)})$. Since node 0 writes $S_0 = 2n \mod K$ at $t_{0(2n)}$, we have $r_0 = 2n-1 \mod K$ at $t_{0(2n)}$. Hence at $t^+_{0(2n)}$, $S_1 = S_2 = \cdots = S_{n-1} = r_0 = r_1 = \cdots = r_{n-1} = 2n-1 \mod K$, and $S_0 = 2n \mod K$. Therefore the system reaches a legitimate configuration of Type 2 at $t^+_{0(2n)}$. ■

Lemma 3. *After reaching a legitimate configuration of Type 2, the system will stay in legitimate configurations forever, and each node in the system will be privileged infinitely many times.*

Proof. (1) If, at a certain instant, the system is in a legitimate configuration of Type 2, then the next move in the system must be node i reading $r_i = S_{i-1} = 1$. Hence the system configuration will change to a legitimate configuration of Type 3.
(2) If, at a certain instant, the system is in a legitimate configuration of Type 3, then the next move in the system must be node i writing $S_i = 1$. (a) If $i \neq n-1$, then the system configuration will change to a legitimate configuration of Type 2. (b) If $i = n-1$, then the system configuration will change to a legitimate configuration of Type 4.
(3) If, at a certain instant, the system is in a legitimate configuration of Type 4, then the next move in the system must be node 0 reading $r_0 = S_{n-1} = 1$. Hence the system configuration will change to a legitimate configuration of Type 1.
(4) If, at a certain instant, the system is in a legitimate configuration of Type 1, then the next move in the system must be node 0 writing $S_0 = 1$. Hence the system configuration will change to a legitimate configuration of Type 2.

From the above and Lemma 2, one can see that the first claim of the lemma is proved. The second claim of the lemma can be verified by also consulting the above discussion, and the details are thus omitted. ■

From Lemmas 2 and 3, we obtain the following theorem.

Theorem 1. *For $K \geq 2n-1$, Algorithm 1 is self-stabilizing under the Dolev model, and solves the mutual exclusion problem.*

Table 1 exhibits an execution of Algorithm 1 for $n = 5$ and $K = 8$. In each configuration in Table 1, the shaded part indicates the execution of a single atomic step by the unique processor selected by the central demon. One can see that in this execution, configuration 81 is exactly the same as configuration 1 and therefore the execution is to be understood as infinitely cyclic with a period of 80. Since none from configuration 1 to configuration 80 is a legitimate configuration, the execution does not contain any legitimate configuration. Therefore Algorithm 1 is not self-stabilizing for $n = 5$ and $K = 8$. Similarly, for any n and K, with $K < 2n - 1$, it is not difficult to find an execution of Algorithm 1 that contains no legitimate configuration. Hence we have obtained the following theorem.

Theorem 2. *For $K < 2n - 1$, Algorithm 1 is not self-stabilizing under the Dolev model.*

4. Quasi-self-stabilization of Algorithm 1

In order to claim that the system equipped with Algorithm 1 is quasi-self-stabilizing under the Dolev model, and solves the mutual exclusion problem, we are required to show that starting with any 0-configuration, Algorithm 1 can ensure that (1) the system eventually converges to a legitimate configuration and then stays in legitimate configurations forever (thus there will always be no more than one node privileged); and (2) every node in the system will be privileged infinitely many times. Note that as mentioned previously, if a system starts with a 0-configuration, then every node in the system will execute its own program from the very first instruction. Therefore in any execution of Algorithm 1 that starts with a 0-configuration, every node in the system will execute a read operation first.

Lemma 4. *If $K \geq n$, then in any execution of Algorithm 1 that starts with a 0-configuration, the system will eventually converge to a legitimate configuration of Type 2.*

Proof. Let 0 be the initial time instant. Without loss of generality, we assume that $S_0 = 0$ at 0. Since node 0 can write infinitely many times in view of Lemma 1, we let $t_{01}, t_{02}, \ldots, t_{0n}$, with $0 < t_{01} < t_{02} < \cdots < t_{0n}$, be the first n instants at which node 0 executes write operations so that S_0 changes to $1, 2, \ldots, n, \mod K$, in sequence. Thus at the instants $t_{01}, t_{02}, \ldots, t_{0n}$, $r_0 = 0, 1, \ldots, n-1$, in sequence. Hence there exist $v_1, v_2, \ldots, v_n$, with $0 < v_1 < t_{01} < v_2 < t_{02} < \cdots < t_{0(n-1)} < v_n < t_{0n}$, such that at the instants $v_1, v_2, \ldots, v_n$, node 0 executes read operations so that r_0 changes to $0, 1, 2, \ldots, n-1, \mod K$, in sequence, and hence at the instants $v_1, v_2, \ldots, v_n$, $S_{n-1} = 0, 1, \ldots, n-1, \mod K$, in sequence. (Note that the existence of the time instant v_1 is due to node 0 executing a read operation first.) Hence there exist $u_2, u_3, \ldots, u_n$, with $v_1 < u_2 < v_2 < \cdots < v_{n-1} < u_n < v_n$, such that at the instants $u_2, u_3, \ldots, u_n$, node $n-1$ executes write operations so that S_{n-1} changes to $1, 2, \ldots, n-1, \mod K$, in sequence. Thus there exist $t_{(n-1)2}, t_{(n-1)3}, \ldots, t_{(n-1)n}$, with $0 < t_{(n-1)2} < t_{(n-1)3} < \cdots < t_{(n-1)n} < t_{0n}$, such that at the instants $t_{(n-1)2}, t_{(n-1)3}, \ldots, t_{(n-1)n}$, node $n-1$ executes write operations so that S_{n-1} changes to $1, 2, \ldots, n-1, \mod K$, in sequence (here, simply let $t_{(n-1)j} = u_j$). Hence at the instants $t_{(n-1)2}, t_{(n-1)3}, \ldots, t_{(n-1)n}$, $r_{n-1} = 1, 2, \ldots, n-1, \mod K$, in sequence. Therefore there exist $q_2, q_3, \ldots, q_n$, with $0 < q_2 < t_{(n-1)2} < q_3 < t_{(n-1)3} < \cdots < t_{(n-1)(n-1)} < q_n < t_{(n-1)n}$, such that at the instants $q_2, q_3, \ldots, q_n$, node $n-1$ executes read operations so that r_{n-1} changes to $1, 2, \ldots, n-1, \mod K$, in sequence, and hence at the instants $q_2, q_3, \ldots, q_n$, $S_{n-2} = 1, 2, \ldots, n-1, \mod K$, in sequence. (Again, the existence of q_2 is due to node $n-1$ executing a read operation first.) Hence there exist $l_3, l_4, \ldots, l_n$, with $q_2 < l_3 < q_3 < \cdots < q_{n-1} < l_n < q_n$, such that at the instants $l_3, l_4, \ldots, l_n$, node $n-2$ executes write operations so that S_{n-2} changes to $2, 3, \ldots, n-1, \mod K$, in sequence. Thus there exist $t_{(n-2)3}, t_{(n-2)4}, \ldots, t_{(n-2)n}$, with $0 < t_{(n-2)3} < t_{(n-2)4} < \cdots < t_{(n-2)n} < t_{(n-1)n}$, such that at the instants $t_{(n-2)3}, t_{(n-2)4}, \ldots, t_{(n-2)n}$, node $n-2$ executes write operations so that S_{n-2} changes to $2, 3, \ldots, n-1, \mod K$, in sequence (here, simply let $t_{(n-2)j} = l_j$).

Arguing in the same manner as above, we will finally get $t_{1n}, t_{2n}, \ldots, t_{(n-1)n}$, with $0 < t_{1n} < t_{2n} < \cdots < t_{(n-1)n} < t_{0n}$, such that for any $i = 1, 2, \ldots, n-1$, node i executes a write operation at t_{in} so that S_i changes to $n-1$. Since node 1 executes a write operation at t_{1n} so that x_1 changes to $n-1$, we have $r_1 = n-1$ at t_{1n}. Hence there exists a t', with $0 < t' < t_{1n}$, such that at the instant t', node 1 executes a read operation so that r_1 changes to $n-1$, and hence at the instant t', $S_0 = n-1$. (Here again, the existence of t' is due to node 1 executing a read operation first.)

Summarizing all the above, we have

(1) $0 < t' < t_{1n} < t_{2n} < \cdots < t_{(n-1)n} < t_{0n}$
(2) $S_0 = n-1$ at t'
(3) $0, 1, \ldots, n-1, \mod K$, are all distinct (since $K \geq n$ by assumption)
(4) $S_0 = 0, 1, \ldots, n-1$ in $(0, t_{01}), (t_{01}, t_{02}), \ldots, (t_{0(n-1)}, t_{0n})$, in sequence (by the definitions of $t_{01}, t_{02}, \ldots, t_{0n}$)
(5) For any $i = 1, 2, \ldots, n-1$, node i executes a write operation at t_{in} so that S_i changes to $n-1$, and hence $r_i = n-1$ at t_{in}
(6) $S_0 = n-1$ in $(t_{0(n-1)}, t_{0n})$. (We single this condition out from those in (4) above for presentation's sake.)

From (1), (2), (3) and (4), we get $t' \in (t_{0(n-1)}, t_{0n})$, and hence $t_{0(n-1)} < t' < t_{1n} < t_{2n} < \cdots < t_{(n-1)n} < t_{0n}$. This, together with (5) and (6), implies $S_1 = r_1 = n-1$ in (t_{1n}, t_{0n}), $S_2 = r_2 = n-1$ in $(t_{2n}, t_{0n}), \ldots,$ and $S_{n-1} = r_{n-1} = n-1$ in $(t_{(n-1)n}, t_{0n})$, in view of Algorithm 1. Thus we have $S_1 = S_2 = \cdots = S_{n-1} = r_1 = r_2 = \cdots = r_{n-1} = n-1$ in $(t_{(n-1)n}, t_{0n})$. Since node 0 writes $S_0 = n \bmod K$ at t_{0n}, we have $r_0 = n-1$ at t_{0n}. Hence at t_{0n}^+, $S_1 = S_2 = \cdots = S_{n-1} = r_0 = r_1 = \cdots = r_{n-1} = n-1$, and $S_0 = n \bmod K$. Therefore the system reaches a legitimate configuration of Type 2 at t_{0n}^+. ■

Table 1
An execution of Algorithm 1 for $n = 5$ and $K = 8$ that does not contain any legitimate configuration.

Configuration number	Node identity									
	0		1		2		3		4	
	r_0	S_0	r_1	S_1	r_2	S_2	r_3	S_3	r_4	S_4
1	0	0	0	7	6	5	4	3	2	1
2	0	1	0	7	6	5	4	3	2	1
3	1	1	0	7	6	5	4	3	2	1
4	1	1	0	7	6	5	4	3	2	2
5	1	1	0	7	6	5	4	3	3	2
6	1	1	0	7	6	5	4	4	3	2
7	1	1	0	7	6	5	5	4	3	2
8	1	1	0	7	6	6	5	4	3	2
9	1	1	0	7	7	6	5	4	3	2
10	1	1	0	0	7	6	5	4	3	2
11	1	1	1	0	7	6	5	4	3	2
12	1	2	1	0	7	6	5	4	3	2
13	2	2	1	0	7	6	5	4	3	2
14	2	2	1	0	7	6	5	4	3	3
15	2	2	1	0	7	6	5	4	4	3
16	2	2	1	0	7	6	5	5	4	3
17	2	2	1	0	7	6	6	5	4	3
18	2	2	1	0	7	7	6	5	4	3
19	2	2	1	0	0	7	6	5	4	3
20	2	2	1	1	0	7	6	5	4	3
21	2	2	2	1	0	7	6	5	4	3
22	2	3	2	1	0	7	6	5	4	3
23	3	3	2	1	0	7	6	5	4	3
24	3	3	2	1	0	7	6	5	4	4
25	3	3	2	1	0	7	6	5	5	4
26	3	3	2	1	0	7	6	6	5	4
27	3	3	2	1	0	7	7	6	5	4
28	3	3	2	1	0	0	7	6	5	4
29	3	3	2	1	1	0	7	6	5	4
30	3	3	2	2	1	0	7	6	5	4
31	3	3	3	2	1	0	7	6	5	4
32	3	4	3	2	1	0	7	6	5	4
33	4	4	3	2	1	0	7	6	5	4
34	4	4	3	2	1	0	7	6	5	5
35	4	4	3	2	1	0	7	6	6	5
36	4	4	3	2	1	0	7	7	6	5
37	4	4	3	2	1	0	0	7	6	5
38	4	4	3	2	1	1	0	7	6	5
39	4	4	3	2	2	1	0	7	6	5
40	4	4	3	3	2	1	0	7	6	5
41	4	4	4	3	2	1	0	7	6	5
42	4	5	4	3	2	1	0	7	6	5
43	5	5	4	3	2	1	0	7	6	5
44	5	5	4	3	2	1	0	7	6	6
45	5	5	4	3	2	1	0	7	7	6
46	5	5	4	3	2	1	0	0	7	6

Table 1 (*continued*)

Configuration number	Node identity									
	0		1		2		3		4	
	r_0	S_0	r_1	S_1	r_2	S_2	r_3	S_3	r_4	S_4
47	5	5	4	3	2	1	1	0	7	6
48	5	5	4	3	2	2	1	0	7	6
49	5	5	4	3	3	2	1	0	7	6
50	5	5	4	4	3	2	1	0	7	6
51	5	5	5	4	3	2	1	0	7	6
52	5	6	5	4	3	2	1	0	7	6
53	6	6	5	4	3	2	1	0	7	6
54	6	6	5	4	3	2	1	0	7	7
55	6	6	5	4	3	2	1	0	0	7
56	6	6	5	4	3	2	1	1	0	7
57	6	6	5	4	3	2	2	1	0	7
58	6	6	5	4	3	3	2	1	0	7
59	6	6	5	4	4	3	2	1	0	7
60	6	6	5	5	4	3	2	1	0	7
61	6	6	6	5	4	3	2	1	0	7
62	6	7	6	5	4	3	2	1	0	7
63	7	7	6	5	4	3	2	1	0	7
64	7	7	6	5	4	3	2	1	0	0
65	7	7	6	5	4	3	2	1	1	0
66	7	7	6	5	4	3	2	2	1	0
67	7	7	6	5	4	3	3	2	1	0
68	7	7	6	5	4	4	3	2	1	0
69	7	7	6	5	5	4	3	2	1	0
70	7	7	6	6	5	4	3	2	1	0
71	7	7	7	6	5	4	3	2	1	0
72	7	0	7	6	5	4	3	2	1	0
73	0	0	7	6	5	4	3	2	1	0
74	0	0	7	6	5	4	3	2	1	1
75	0	0	7	6	5	4	3	2	2	1
76	0	0	7	6	5	4	3	3	2	1
77	0	0	7	6	5	4	4	3	2	1
78	0	0	7	6	5	5	4	3	2	1
79	0	0	7	6	6	5	4	3	2	1
80	0	0	7	7	6	5	4	3	2	1
81	0	0	0	7	6	5	4	3	2	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

From Lemmas 3 and 4, we obtain the following theorem.

Theorem 3. *For $K \geq n$, Algorithm 1 is quasi-self-stabilizing under the Dolev model, and solves the mutual exclusion problem.*

Table 2 exhibits an execution of Algorithm 1 for $n = 7$ and $K = 6$ that starts with a 0-configuration. Note that in this execution, every node executes a read operation first. In each configuration in Table 2, the shaded part indicates the execution of a single atomic step by the unique processor selected by the central demon. One can see that in this execution, configuration 85 is exactly the same as configuration 1, and therefore the execution is to be understood as infinitely cyclic with a period of 84. Since none from configuration 1 to configuration 84 is a legitimate configuration, the execution does not contain any legitimate configuration. Therefore Algorithm 1 is not quasi-self-stabilizing for $n = 7$ and $K = 6$. Similarly, for any n and K, with $K < n$, it is not difficult to find an execution of Algorithm 1 which starts with a 0-configuration, and contains no legitimate configuration. Hence we have obtained the following theorem.

Table 2
An execution of Algorithm 1 for $n = 7$ and $K = 6$ that starts with a 0-configuration and contains no legitimate configuration.

Configuration number	Node identity													
	0		1		2		3		4		5		6	
	r_0	S_0	r_1	S_1	r_2	S_2	r_3	S_3	r_4	S_4	r_5	S_5	r_6	S_6
1	5	0	5	5	4	4	3	3	2	2	1	1	0	0
2	0	0	5	5	4	4	3	3	2	2	1	1	0	0
3	0	0	5	5	4	4	3	3	2	2	1	1	1	0
4	0	0	5	5	4	4	3	3	2	2	1	1	1	1
5	0	0	5	5	4	4	3	3	2	2	2	1	1	1
6	0	0	5	5	4	4	3	3	2	2	2	2	1	1
7	0	0	5	5	4	4	3	3	3	2	2	2	1	1
8	0	0	5	5	4	4	3	3	3	3	2	2	1	1
9	0	0	5	5	4	4	4	3	3	3	2	2	1	1
10	0	0	5	5	4	4	4	4	3	3	2	2	1	1
11	0	0	5	5	5	4	4	4	3	3	2	2	1	1
12	0	0	5	5	5	5	4	4	3	3	2	2	1	1
13	0	0	0	5	5	5	4	4	3	3	2	2	1	1
14	0	0	0	0	5	5	4	4	3	3	2	2	1	1
15	0	1	0	0	5	5	4	4	3	3	2	2	1	1
16	1	1	0	0	5	5	4	4	3	3	2	2	1	1
17	1	1	0	0	5	5	4	4	3	3	2	2	2	1
18	1	1	0	0	5	5	4	4	3	3	2	2	2	2
19	1	1	0	0	5	5	4	4	3	3	3	2	2	2
20	1	1	0	0	5	5	4	4	3	3	3	3	2	2
21	1	1	0	0	5	5	4	4	4	3	3	3	2	2
22	1	1	0	0	5	5	4	4	4	4	3	3	2	2
23	1	1	0	0	5	5	5	4	4	4	3	3	2	2
24	1	1	0	0	5	5	5	5	4	4	3	3	2	2
25	1	1	0	0	0	5	5	5	4	4	3	3	2	2
26	1	1	0	0	0	0	5	5	4	4	3	3	2	2
27	1	1	1	0	0	0	5	5	4	4	3	3	2	2
28	1	1	1	1	0	0	5	5	4	4	3	3	2	2
29	1	2	1	1	0	0	5	5	4	4	3	3	2	2
30	2	2	1	1	0	0	5	5	4	4	3	3	2	2
31	2	2	1	1	0	0	5	5	4	4	3	3	3	2
32	2	2	1	1	0	0	5	5	4	4	3	3	3	3
33	2	2	1	1	0	0	5	5	4	4	4	3	3	3
34	2	2	1	1	0	0	5	5	4	4	4	4	3	3
35	2	2	1	1	0	0	5	5	5	4	4	4	3	3
36	2	2	1	1	0	0	5	5	5	5	4	4	3	3
37	2	2	1	1	0	0	0	5	5	5	4	4	3	3
38	2	2	1	1	0	0	0	0	5	5	4	4	3	3
39	2	2	1	1	1	0	0	0	5	5	4	4	3	3
40	2	2	1	1	1	1	0	0	5	5	4	4	3	3
41	2	2	2	1	1	1	0	0	5	5	4	4	3	3
42	2	2	2	2	1	1	0	0	5	5	4	4	3	3
43	2	3	2	2	1	1	0	0	5	5	4	4	3	3
44	3	3	2	2	1	1	0	0	5	5	4	4	3	3
45	3	3	2	2	1	1	0	0	5	5	4	4	4	3
46	3	3	2	2	1	1	0	0	5	5	4	4	4	4

Table 2 (*continued*)

Configuration number	Node identity													
	0		1		2		3		4		5		6	
	r_0	S_0	r_1	S_1	r_2	S_2	r_3	S_3	r_4	S_4	r_5	S_5	r_6	S_6
47	3	3	2	2	1	1	0	0	5	5	5	4	4	4
48	3	3	2	2	1	1	0	0	5	5	5	5	4	4
49	3	3	2	2	1	1	0	0	0	5	5	5	4	4
50	3	3	2	2	1	1	0	0	0	0	5	5	4	4
51	3	3	2	2	1	1	1	0	0	0	5	5	4	4
52	3	3	2	2	1	1	1	1	0	0	5	5	4	4
53	3	3	2	2	2	1	1	1	0	0	5	5	4	4
54	3	3	2	2	2	2	1	1	0	0	5	5	4	4
55	3	3	3	2	2	2	1	1	0	0	5	5	4	4
56	3	3	3	3	2	2	1	1	0	0	5	5	4	4
57	3	4	3	3	2	2	1	1	0	0	5	5	4	4
58	4	4	3	3	2	2	1	1	0	0	5	5	4	4
59	4	4	3	3	2	2	1	1	0	0	5	5	5	4
60	4	4	3	3	2	2	1	1	0	0	5	5	5	5
61	4	4	3	3	2	2	1	1	0	0	0	5	5	5
62	4	4	3	3	2	2	1	1	0	0	0	0	5	5
63	4	4	3	3	2	2	1	1	1	0	0	0	5	5
64	4	4	3	3	2	2	1	1	1	1	0	0	5	5
65	4	4	3	3	2	2	2	1	1	1	0	0	5	5
66	4	4	3	3	2	2	2	2	1	1	0	0	5	5
67	4	4	3	3	3	2	2	2	1	1	0	0	5	5
68	4	4	3	3	3	3	2	2	1	1	0	0	5	5
69	4	4	4	3	3	3	2	2	1	1	0	0	5	5
70	4	4	4	4	3	3	2	2	1	1	0	0	5	5
71	4	5	4	4	3	3	2	2	1	1	0	0	5	5
72	5	5	4	4	3	3	2	2	1	1	0	0	5	5
73	5	5	4	4	3	3	2	2	1	1	0	0	0	5
74	5	5	4	4	3	3	2	2	1	1	0	0	0	0
75	5	5	4	4	3	3	2	2	1	1	1	0	0	0
76	5	5	4	4	3	3	2	2	1	1	1	1	0	0
77	5	5	4	4	3	3	2	2	2	1	1	1	0	0
78	5	5	4	4	3	3	2	2	2	2	1	1	0	0
79	5	5	4	4	3	3	3	2	2	2	1	1	0	0
80	5	5	4	4	3	3	3	3	2	2	1	1	0	0
81	5	5	4	4	4	3	3	3	2	2	1	1	0	0
82	5	5	4	4	4	4	3	3	2	2	1	1	0	0
83	5	5	5	4	4	4	3	3	2	2	1	1	0	0
84	5	5	5	5	4	4	3	3	2	2	1	1	0	0
85	5	0	5	5	4	4	3	3	2	2	1	1	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Theorem 4. *For $K < n$, Algorithm 1 is not quasi-self-stabilizing under the Dolev model.*

5. Concluding remarks

In the above, we have introduced the new notion of quasi-self-stabilizing systems that reflects an interesting phenomenon observed in our research on self-stabilizing systems of the Dolev type. The phenomenon was observed when

we tried to use the approach taken in this paper to study the self-stabilization property of a naturally-adapted version (Algorithm 1) of Dijkstra's K-state mutual exclusion algorithm for a ring network. We have obtained the following two main results, which can clarify the new notion:

(1) Algorithm 1 is self-stabilizing under the Dolev model if and only if $K \geq 2n - 1$ (Theorems 1 and 2), and
(2) Algorithm 1 is quasi-self-stabilizing under the Dolev model if and only if $K \geq n$ (Theorems 3 and 4).

To the best of our knowledge, the first main result has only been proven in Dolev's book [8] in the past. As mentioned earlier, the main idea of Dolev's proof is by transforming a problem under the read/write atomicity model into a problem under the central demon model. Our proof for the same result, using the same approach taken in [3,10], deals directly with the complexity resulting from the refined read/write atomicity. Mainly due to such a difference, our proof can be modified to obtain the quasi-self-stabilization property of Algorithm 1, the second main result of this paper.

References

[1] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM 17 (1974) 643–644.
[2] L. Lamport, Solved problems, unsolved problems and non-problems in concurrency, invited address, in: Proc. of the Third Annual ACM Symposium on Principles of Distributed Computing, 1984 pp. 1–11.
[3] T.C. Huang, A self-stabilizing algorithm for the shortest path problem assuming read/write atomicity, J. Comput. Syst. Sci. 71 (2005) 70–85.
[4] T.C. Huang, A self-stabilizing algorithm for the shortest path problem assuming the distributed demon, Comput. Math. Appl. 50 (2005) 671–681.
[5] E.W. Dijkstra, Self-stabilization in spite of distributed control, in: Selected Writings on Computing a Personal Perspective, Springer-Verlag, Berlin, Heidelberg, New York, 1982, pp. 41–46.
[6] E.W. Dijkstra, A belated proof of self-stabilization, Distrib. Comput. 1 (1986) 5–6.
[7] S. Dolev, A. Israeli, S. Moran, Self-stabilization of dynamic systems assuming only read/write atomicity, Distrib. Comput. 7 (1993) 3–16.
[8] S. Dolev, Self-Stabilization, MIT Press, Massachusetts, 2000.
[9] Z. Collin, S. Dolev, Self-stabilizing depth-first search, Inform. Process. Lett. 49 (1994) 297–301.
[10] T.C. Huang, J.C. Lin, N. Mou, A self-stabilizing algorithm for the center-finding problem assuming read/write atomicity, Comput. Math. Appl. 48 (2004) 1667–1676.
[11] S. Shukla, D.J. Rosenkrantz, S.S. Ravi, Observations on self-stabilizing graph algorithms on anonymous networks, in: Proc. of the 2nd Workshop on Self-Stabilizing Systems, WSS, 1995, pp. 7.1–7.15.